WOI'00: New Issues in Object Interoperability

Antonio Vallecillo[1], Juan Hernández[2], and José M. Troya[1]

[1] Universidad de Málaga, Spain
{av,troya}@lcc.uma.es
[2] Universidad de Extremadura, Spain
juanher@unex.es

Abstract. This report summarizes the presentations, discussions, and outcomes of the ECOOP'2000 Workshop on Object Interoperability, held in Sophia Antipolis, France, on Monday, June 12, 2000. Divided into four main sessions, the workshop covered some of the most important issues related to object interoperability at different levels (such as protocols or semantics). This report has two main goals. First, it tries to provide a snapshot of some of the current research being carried out within the object-oriented community in these areas. And second, it summarizes some of the open questions and issues related to object interoperability, in order to set the basis for further research activities.

1 Introduction

In the Open Systems arena there is now a race under way between middleware architects and vendors to establish interoperational standards (e.g. CORBA, EJB or DCOM), as well as bridges among them. Those component platforms undoubtedly provide the infrastructural support for components to interoperate at certain (basic) levels. However, much work remains to be done by the object-orientation community in order to deal with all the complex aspects that object interoperability embraces in open systems. In this context, interoperability can be defined as the ability of two or more entities to communicate and cooperate despite differences in the implementation language, the execution environment or the model abstraction [9].

In order to discuss some of the issues related to object interoperability, the first ECOOP Workshop on Object Interoperability (WOI'99) was organized in association with the 13th European Conference on Object-Oriented Programming (ECOOP'99), and held in Lisbon (Portugal) in June 1999. WOI'99 successfully contributed to gather a number of researchers interested in object interoperability, and allowed them to start building a common understanding of the different problems and yet unexplored areas to be investigated. Basically, three main levels of interoperability between objects were distinguished at WOI'99: the *signature level* (names and signatures of operations), the *protocol level* (relative order between exchanged messages and blocking conditions), and the *semantic level* ("meaning" of operations).

Although interoperability is currently well defined and understood at the signature level, this sort of interoperability is not sufficient for ensuring the

correct development of applications in open systems. Typical interface definition languages (IDLs) provide just the syntactic descriptions of the objects' public methods, i.e. their signatures. However, nothing is said about the order in which the objects expect their methods to be called, their blocking conditions, or their functionality. Basically, current IDLs do not describe the usage, requirements, capabilities, and behavior of objects.

The variety of topics covered during WOI'99 revealed the wide range of challenges that the study of object interoperability at both protocol and semantic levels brings out when objects have to interoperate in open systems. The list of questions and open issues was so long that we decided to organize a new edition of the object interoperability workshop, with the aim of promoting research concerned with all aspects of interoperability between objects, in particular at the protocol and semantic levels.

2 The WOI'00 Workshop

The second Workshop on Object Interoperability (WOI'00) was held in Sophia Antipolis, France, in conjunction with ECOOP'2000. The main aim of WOI'00 was to provide a venue where researchers and practitioners concerned with all aspects of interoperability could meet, disseminate and exchange ideas and problems, identify some of the key issues related to object interoperability, and explore possible solutions. As a WOI'00 new feature, special attention was drawn into some of the existing commercial object models, proposing extensions of their IDLs in order to cope with some of those interoperability issues, and discussing how to check them during compilation and run time in commercial applications. In particular, the topics of interest pointed out in the call for papers included, among others:

- Extensions to object interfaces and IDLs to deal with protocol or semantic interoperability (especially commercial IDLs).
- Enabling models, technologies and architectures for object interoperability.
- Protocol and semantic checking techniques.
- Industrial and experience reports.

All submitted papers were formally reviewed by at least two referees, and 10 papers were finally accepted for presentation at the workshop. Contributions were divided into four distinct groups:

1. In the first group we have the contributions describing *experiences with commercial object models*, encouraging the need of formalizing CORBA service IDLs and specifications.
2. In the second group we find those focusing on *protocol interoperability*, including IDL extensions, protocol checking techniques, and some formal aspects of interoperability at this level.
3. Interesting experience reports about *syntactic interoperability* are found in the third group of contributions.

4. Finally, two contributions propose *interaction frameworks* for object interoperability.

In addition, this year we had a brilliant keynote speech by Prof. Mehmet Aksit, from the University of Twente, that opened the workshop. His talk served not only for introducing the topics related to object interoperability and the key concepts behind them, but also for setting up a common vocabulary and understanding among participants. There were many references to the concepts and the notations introduced by Prof. Aksit during the debates, which proves the invaluable help that having a keynote speaker represents. In particular, his talk significantly helped bringing all participants into a shared conceptual framework on which to base all discussions.

The workshop gathered 13 people from 7 different countries, who actively participated in very lively discussions. Their names and affiliations are listed in Annex 1. This report has been put together with the help of all those participants, and summarizes the workshop presentations, discussions, and results obtained. It is structured in 5 sections. After this introduction, section 3 describes the first three sessions, starting with a summary of the keynote speech, and the outcomes of the paper presentations and debates. Section 4 is dedicated to the last session, and contains some of the issues that were finally raised for discussion, the actions identified as post-workshop homework, and the final conclusions. This section also includes the discussions that took place after the workshop, during the production of this chapter, and the conclusions agreed.

In addition to this report, a book with the Proceedings of the workshop has been produced [4], which contains the full-length version of all selected papers. More information about the workshop and its participants is also available at the workshop Web page: `//webepcc.unex.es/juan/woi00/`.

3 Presentations

The workshop was divided into 4 sessions (two in the morning, two in the afternoon). This section describes the first three, which were devoted to discussing the presented papers. A brief outline of the ideas of the keynote speech and each presented paper is given, as well as some of the conclusions reached after each one. Those conclusions were written down in the blackboard, and finally summarized during the last session.

3.1 Keynote Speech: Quality-aware Interoperability (Mehmet Aksit)

Nature evolution provides a good analogy for studying how (successful) products and systems should be built: every successful design of a complex system requires a *proper decomposition*, usually based on a solution domain knowledge. Similarly, every successful product displays a *proper mix of quality factors*, and is composed of well-balanced cooperating structures. Besides, every product must survive in

a competitive environment, and hence it should *adapt* to seamlessly cooperate with other products, and with its environment.

Proper decomposition should be driven by balancing quality factors, specially those imposed by the users' requirements such as functionality, reusability, adaptability, or performance. So, software systems should be decomposed in such a way that the right mix of those factors is achieved.

One effective way of reasoning about composition is by using the formula $S_3 = S_2 \oplus S_1$, in which S_1 is what we currently have, S_3 is what we need, S_2 represents what we need to add, and operator $\oplus$ models how we should add it. This expression allows to represent the different ways in which software systems are built. For instance, in an *extension* S_1 exists but S_2 is yet to be implemented. In *composition* both S_1 and S_2 exist, and S_3 has to be defined by composing S_2 and S_1. In *run-time adaptation* the operator $\oplus$ can be provided by the system and applied at run-time, while in *compile-time adaptation* the operator $\oplus$ can be expressed within the adopted programming language and corresponds to writing "glue-code" (let it be inheritance or coordination, for instance). In addition, this formula also allows to *measure* the various quality factors, evaluating the different costs of reusability, adaptability, etc.

On the other hand, cooperation requires compatibility and synchronization. In order to accomplish software compatibility, two main levels can be distinguished —procedural and functional—, which deal respectively with syntactic and semantic agreements. Three are the main requirements for functional compatibility: relevance, semantic compatibility, and realizability. Relevance is the most basic requirement, and means that both S_2 and S_1 must be relevant for defining S_3. Semantic compatibility refers to the semantic consistency of the definitions of both S_2 and S_1 for defining S_3. Finally, realizability requirements cover the compatibility restrictions for the realization of the services and composition mechanisms (e.g., lack of expression power, extensibility or adaptability).

Relevancy must be determined within the context of a certain business scenario, and can be assured by a successful requirement analysis process. In its turn, semantic compatibility needs synthesis and verification. The synthesis process aims at searching solutions from the solution domain and extracting the abstractions from the selected solution domain, while verification cares about having the sub-concerns of S_1 and S_2 compared and verified for semantic compatibility.

Summarizing, interoperability becomes a necessity due to decomposition of systems into sub-systems (say $S_1, S_2, .., S_n$). Decomposition is by itself a compromise because of the necessity to provide a right balance between quality factors, such as functionality, reusability, adaptability, or performance. Interoperability requires proper decomposition based on a solution domain knowledge, and balancing quality factors needs to be the driving force for decomposition.

3.2 Experiences with commercial object models

Two papers were presented in this group, which outline some of the limitations of current CORBA specifications, and propose some ideas to overcome them.

3.2.1 Towards Components that Plug AND Play (R. Bastide, O. Sy)

Just specifying the syntactic level of cooperation with IDLs, and informally describing the behavior of objects does not necessarily lead to fruitful cooperation in an object-oriented system. In this paper the authors advocate that behavioral specifications of objects should be formal if they have to be non-ambiguous. They make their point by providing the conclusions of some interoperability tests conducted on five commercial implementations of the Object Management Group's CORBA Event Service. These tests expose the non-interoperability of some implementations which trace back, on one hand, to the original specification's incompleteness and, on the other hand, to mistakes in the implementation. The authors propose that the first cause be corrected by the use of a formalism suited to the needs of CORBA systems (like the Petri nets based Cooperative Objects formalism [2]), and the second by the provision of test cases with the original specification.

Two main conclusions were drawn from the discussions. Firstly, specifications in natural language are not enough because precision is needed. Specifications should be formal, and test cases should be provided to the user. However, although formal behavioral specifications are needed, they are not enough to ensure object interoperability. It is necessary to make requirements for interoperability explicit (see section 4.2) in order to make a conscious rational choice based on analysis of available design (and later technological) options. Secondly, since different formal specification languages are available and provide different views of interoperability (levels of abstraction, performance, real-time, size, scalability, system modeling, system composition, etc.), there is an issue of selecting and combining these views (see section 4.3). However, in order to obtain this global view, one must consider the trade-offs: *"all qualities cannot be obtained in one product"* (cf. Prof. Aksit's presentation).

3.2.2 Adding Protocol Information to CORBA IDLs (C. Canal, L. Fuentes, J.M. Troya, A.Vallecillo)

Traditional IDLs were defined for describing the services that objects offer, but not those services they require from other objects, nor the relative order in which they expect their methods to be used. In this paper the authors propose an extension of the CORBA IDL that uses a sugared subset of the polyadic π-calculus for describing object service protocols, aimed at the automated checking of protocol interoperability between CORBA objects in open component-based environments.

Once it is shown how CORBA objects' interface descriptions can be enriched with protocol information, the authors discuss the sort of checks that can be carried out with their proposal (including some safety and liveness properties of applications, and component compatibility and substitutability checks), as well as some of its advantages and disadvantages. Furthermore, the authors also discuss the practical utility of their proposal, and present the limitations

encountered when trying to implement and use this sort of IDL extensions in open systems.

Two main conclusions were reached after the discussions that followed this presentation. First, that interoperability tests can be costly due to their (intrinsic) complexity. Nevertheless, including knowledge about the solution domain may significantly reduce the complexity of the tests. Secondly, it was also agreed that formal description techniques (FDTs) may not be enough for specifying systems: not all aspects and requirements can be covered by FDTs.

3.3 Protocols

The second session was dedicated to the discussion of interoperability issues at the *protocol* level. In our context, a protocol is a restriction on the ordering of incoming and/or outgoing messages, including possible blocking conditions and guards on the objects' methods. Four papers were selected in this group, covering different proposals that address those issues from very different perspectives.

3.3.1 An Enhanced Model for Component Interfaces to Support Automatic and Dynamic Adaption (Ralf H. Reussner)

Current commercial component systems, modeling component interfaces as a list of method signatures, have several practical relevant shortcomings. For instance, they do not detect component compositional errors before the actual use of the composed system; besides, all adaptations of a component must be explicitly foreseen and programmed by the developer. The author addresses these problems by a new component model for Java: CoCoNut/J. A CoCoNut enhances a Java Bean with a clear COmponent interface COntract. This new interface model is regarded as a contract to deploy a component: the services that the component offers (call-interface) and the external services it requires from other components (use-interface). The call-interface models not only the available services, but also all valid call sequences to these services. Alike, the use-interface models all possible call sequences to used services of an external component.

The linkage between both interfaces is explicitly modeled. When not offering a service in the call-interface we may also not require certain external services. Conversely, if some external services are not available, some of the component's services may not be offered either. The author shows how this connection of the interfaces can be actually used to automatically (re-)compute the component's call-interface with dependency from the existing external services. In the paper the author describes an extension of finite state machines with counters which are used to model the interfaces. With the two interfaces it can be checked during *composition* time (i.e., when the user awaits errors, and *before* run-time) whether two components A and B can work together, i.e., whether the protocol of the use-interface of A fits to the B's protocol of the call-interface.

Generalizing this check, component A can be adapted in the case the protocols do not fit, or component B is missing. The main idea is to exploit the linkage

between the call-interface and the use-interface of component A. In case the use-interface of A does not match the call-interface of B, another use-interface of A can be built which matches the call-interface of B. This is done by constructing the intersection of these interfaces (the cross product of the automata). After having a new use-interface of A which matches the call-interface of B, the new call-interface of A can be generated out of A's automata and its new use-interface. This restriction of functionality is a certain kind of adaptation does not have to be explicitly programmed by the component developer, and is available as a service provided by the component infrastructural system.

3.3.2 On Practical Verification of Processes (Rick van Rein)

When proving properties such as 'compatibility' on processes, the two customary techniques to use are model checking and theorem proving. Model checking is known for its speed and its ability to provide good feedback (to guide repairing faults), but it is limited to systems with a bounded number of process instances. This problem is an important drawback to computer scientific process verification, and for that reason we decided to look into theorem provers.

Theorem provers are powerful proof tools that usually conduct their reasoning at a fair speed. However, one problem of theorem provers lies in the low level of internal reasoning (logic, while a process designer tends to think in terms of process steps) and the resulting feedback (if any); another problem of theorem provers is the danger of infinite loops in the reasoning process.

The problem of the low level of feedback can be solved by clearly distinguishing the *interesting* properties (proof attempts and axioms) from the *cumbersome* ones. A wrapper around a theorem prover formulates desired proofs of *interesting* properties and offers them to a theorem prover as subgoals. Internally, the theorem prover exploits both *interesting* and *cumbersome* knowledge (axioms and previously proven properties) to attempt to achieve such subgoals. The collected successes and failures of these subgoals do not reveal the internal *cumbersome* proof steps, just their end result, which is in terms of *interesting* properties. These successes and failures provide the same information as collected by a model checker, hence the same useful feedback can be provided.

The problem of infinite looping during the reasoning about proofs can be avoided by constraints that define a reasonable class of *acceptable* problems. This class of *acceptable* problems cannot be defined for logic in general, but the use of knowledge from the problem domain (processes) can help out. Consider that processes are associated (they refer to each other) and one way to avoid infinite proofs would be to avoid infinitely cycling through the process associations. This can be avoided by constraining the number of introductions of the quantor rules that describe these associations, and/or the number of introductions of the quantor rules that describe the processes themselves. Such an upper limit to quantor-rule introductions is a well-known solution to the danger of infinite looping.

In conclusion, this paper presents an approach to prove process-related properties with theorem provers, in such a way that good feedback can be given and proofs always consume a finite time.

3.3.3 Temporal Logic Based Specifications of Component Interaction Protocols (Jun Han)

The interaction protocols of software components are critical to their proper understanding and use. Based on real-world experience in designing a telecommunications system, the author presents a temporal logic based approach to the specification of component interaction protocols. The protocol specifications take the form of interaction constraints on a component's required and provided signature elements (i.e., attributes, operations and events).

After giving an overview of a framework for comprehensive and yet flexible component interface specification, the paper introduces a number of temporal constructs for specifying interaction constraints. Then, the types and placement of interaction constraints in the interface specification are analyzed. The paper also provides a comparative discussion with other protocol specification approaches, including those based on description logics, Petri nets, state machines and process algebras. The author finds that the temporal logic based approach in the form of constraints is particularly practical as it is easy for practitioners to learn and use, and it allows *incremental* specification of interaction protocols.

3.3.4 A Framework for the Specification and Testing the Interoperation Aspects of Components (I. Cho)

This work proposes a component specification model that adds a protocol and behavioral aspects of a component to enhance the interoperability between components interacting to each other. The notation used for the specification is an extension of the OCL (object constraint language), which is a part of UML. One notable aspect of the work is the testing framework that integrates the specification model, the syntax/protocol checker, the automatic test sequence and test code generator. The component specification model is an extension of the current industrial standards —CORBA, COM/DCOM, JavaBeans—, and aims at providing their components with more powerful interoperability tests.

3.4 Syntactic Interoperability

In this session two papers describe interesting experience reports about interoperability at the signature level.

3.4.1 A Multi-Level Approach to Verifiable Correct Design and Construction of Multi-language Software (A. Kaplan and J.C. Wileden)

The work by Alan Kaplan and Jack C. Wileden propose an approach for making heterogeneous white-box components interoperate, defending the "interface

bridging" approach instead of the "interface standardization" approach that uses IDLs [5]. In particular, the work presented allows programs written in CLOS and C++ to transparently interoperate by means of a tool called PolySPINner.

An interesting discussion about the real use of formal description techniques followed this presentation. It is clear to all that formal description techniques are far from being widely accepted and utilized in real industrial environments. In fact, type checking is the only practical, useful and accepted way of formal analysis nowadays. Furthermore, type definition is the practical limit which real users are willing to specify (and have reasonable chance of getting right). Several reasons can be argued for that:

1. Semantic specifications are very difficult to write, and the computational complexity of their tests makes them impractical in most situations. Besides, we cannot forget the necessities and abilities of users to whom formal description techniques may be useless. In practice, the more specialized/powerful the formal model, the worse the ratio pay-off/pain.
2. Formal models (even types) always miss the aspects that turn out to matter in the real world, since they do not allow to capture many of the non-functional requirements involved in any real application.
3. And finally, protocol specifications are seldom really useful or sufficiently precise, as previously pointed out.

3.4.2 Interoperability Oviedo3/COM Objects through Automation (F. Dominguez and J.M. Cueva)

In this paper the authors present an experiment of integrating a proprietary object-oriented system (Oviedo3) with COM objects by means of wrapping the `IDispatch` interface. This technique allows COM objects to be perceived and used as native objects by the user, allowing an easy integration between both systems.

3.5 Interaction Frameworks for Object Interoperability

The two last papers dealt with interaction frameworks for object interoperability.

3.5.1 Addressing Interoperability in Multi-Organizational Web-Based Systems (A. Ruiz et al.)

Current techniques for checking interoperability present a number of drawbacks that make it difficult to apply them in nowadays software industry. In this paper the authors identify some of these drawbacks, and present a proposal to tackle object interoperability from a new point of view: interoperability certificates a component must have been granted in order to participate in a collaboration with other components. Certificates are granted when a component passes a number of tests, and they are stored in a repository so that checking interoperability is a simple, quick process.

The authors also rise some questions in order to address the interoperability problem in the context of MOWS (Multi–Organizational Web–Based Systems) that are specially attractive in electronic commerce. MOWS have a number of features that transform them into special open distributed systems where the approach proposed by the authors may be realistic and attractive to software engineers.

Finally, two open issues were risen: the need of a new interoperability level (the *quality level*), and the future of the certification techniques for checking interoperability.

3.5.2 Interaction Framework for Interoperability and Behavioral Analysis (J. Putman and D. Hybertson)

An interaction framework based on a software architecture perspective was presented in terms of a set of views that address issues of interoperability and semantic behavior. The framework uses concepts from the Reference Model for Open Distributed Processing (RM-ODP, an ISO standard). The views represent levels of abstraction, as a means of supporting distributed system interoperability. The architectural concept of connector provides the locus of relations, interactions, and protocols among components. The following views were defined:

- The *relationship view* specifies that a relation exists between objects or components, and defines the interaction.
- The *interface view* specifies the interfaces of each component, using a component model that allows multiple interfaces per component, within a taxonomy of interface types. This view still hides distribution of components.
- The *binding view* enables communication and interaction by focusing on connectors. It is a more concrete view in that it reveals distribution of components.
- The *interceptor view* supports interoperating components in different technical or administrative domains by addressing issues of cross-domain policy management and mediation.
- The *behavioral semantics view* specifies the semantics of the interaction and the information/data involved in the interaction. It supports all the other views.

4 Final Session

The final session was dedicated to discuss some of the issues not covered during the previous sessions, to summarize the conclusions drawn during the presentations into a final list of conclusions, and to select some actions for participants as homework. In addition, a number of discussions happened among the workshop participants by e-mail after the event.

This section is dedicated to present the outcomes of those discussions, as well as the list of the workshop's final conclusions.

4.1 Requirements for Interoperability

In the first place, Duane Hybertson wanted to clarify which were the precise *requirements* for interoperability. The following definition (as prepared by Duane himself) was produced. It groups interoperability requirements according to where the resulting constraints (client side or server side) are located:

Definition 1 (Requirements for Interoperability). *Given two components, 'Client' and 'Server', four general interoperability requirements, or classes of requirements, are proposed:*

R1 Data Constraints. *Data received by the Client from the Server must satisfy all client-side data constraints, including constraints on types, content, units, and quality.*

R2 Services Constraints. *Services received by the Client from the Server must satisfy all client-side service constraints, including functionality (or other relation between input and output such as precondition and postcondition), sequence, timing, and quality of service (QoS).*

R3 Request Constraints. *Services requested by the Client from the Server must satisfy a set of server-side preconditions before they are permitted access to those services. In turn, the Server must satisfy its postcondition (including both functionality and QoS postconditions) in providing the service.*

R4 Control Constraints. *All components (clients and servers) involved in an interaction must have a consistent expectation regarding transfer of control, including blocking conditions.*

4.2 Levels of Interoperability

During the discussions the group did not agree with the original taxonomy of having three levels of object interoperability. Are protocols part of semantics? Are semantics just about behavioral specifications? The following two paragraphs extracted from the e-mail discussions summarize the debate that took place about this taxonomy:

Antonio: ...Yes, we could start arguing about the taxonomy "signatures/protocols/semantics". In this point it could be worth having a look at last year's WOI report [8]. Of course you can stick to the "traditional" signature/semantics taxonomy (also called static/dynamic in [5], or plug/play by Sy and Bastide in the workshop). Even in this case the "semantic" level may include too many things. If we take a look at the literature, most of the authors that say "semantics" mean "operational semantics" or "behavioral specifications". So far people have concentrated in signature interoperability (just method names) and operational semantics (using various formalisms: pre-post, temporal logic, process algebras, etc. —the new book edited by Leavens and Sitaraman compiles many of the approaches that deal with the semantic aspects of components [6]). However, the "semantic" level is too broad. It should cover not only operational semantics and behavioral specifications of components, but also agreements on

names, context-sensitive information, agreements on concepts (ontologies) — which moves names agreements one level up in the meta-data chain—, etc. (All this is very well described in the papers by Sandra Heiler, eg. [3]). The problem is that this is a too broad and general problem to be tackled in full.

On the other hand, dealing with behavioral specifications needs very heavy machinery, which makes unpractical most approaches for real applications. One possibility that was originally proposed by Yellin and Storm in 1994 (inspired by the works by Nierstraz [7], and Allen and Garlan [1]) was to deal just with the "protocol" aspects of the behavior of components. In this way more practical tests could be defined for proving interoperability among components, and actually this has traditionally been the interoperability "level" used in most Architectural Description Languages....

Duane: ...As Antonio has indicated, the concept of "semantics" covers a very broad set of issues, and there is no universal consensus on the full scope of what those issues are. This is not surprising, given the difficulty of the topic. I would simply add that we should be realistic and view "semantics" as a topic that is likely to develop and mature over a lengthy period, perhaps even decades. That is, rather than a sudden revolution in understanding semantics, what may happen is a kind of extended factoring process. Relatively small aspects of the problem will be defined more precisely, solved, and removed from the big, fuzzy bin of what we now call "semantics". The process would be somewhat analogous to the way issues and methods once grouped under "artificial intelligence" are now simply considered useful tools of everyday software engineering. Such a factoring process for semantics could for example explain why we tend to separate semantics from issues such as signatures (which we understand well) and protocols (for which we are gaining a good understanding), even though both of these topics have semantic aspects. As topics are factored out, they may be positioned at different levels, perhaps in something like a lattice structure. Over time the gradual clarification and removal of well-understood topics from the semantics bin will both reduce the size of the semantics problem and help provide a new set of tools for resolving the remaining issues. At the same time, though, I think it is important to preserve an overall understanding of how a thread of semantics is woven throughout all of these areas. That thread begins with topics as "simple" as signatures and extends through issues so abstract that at present we do not even know how to name them. Recognizing this thread of semantics should help us in time to arrive at a deeper and more profound understanding of semantics. Our achievement of this understanding will certainly come in increments, and is not fully articulated by the three-level taxonomy at this time....

4.3 Workshop Conclusions

Finally, Jack Wileden did an excellent job summarizing the workshop conclusions into the following list. Those conclusions represent the ideas agreed by all participants. In addition, this list also points out some of the existing problems that object interoperability currently faces, and provides some hints to their solutions that may be worth investigating.

I. Interoperability is an (increasingly) important concern.
II. Support for interoperability should/must include:
    - Careful specification of the components and their properties.
    - Precise notations of compatibility (composition).
    - Automated tools for constructing, analysing, composing components.
III. Candidate (potentially viable) specification methods include:
    - Signature/types (*sine qua non*, i.e. fundamental).
    - Protocols (partial orders of signature invocations).
    - Behavior (results of sequences of signature invocations).
IV. Underlying formal models are potentially powerful. Valuable specific strengths and weaknesses should continue to be investigated.
V. Underlying formal models should be hidden as thoroughly as possible (both on inputs and output states) from most users.
    - Automatic derivation/inference is a promising possibility.
VI. Other important considerations:
    - Multiple levels of abstractions should be considered.
    - Solution domain (in addition or alternative to implementation domain) driven decomposition.
    - Connections to software architectures (and other related research areas).
    - Possibility of defining patterns for interoperability.

5 Concluding Remarks

If a main goal of a workshop is to gather researchers and practitioners of a discipline, and let them exchange ideas, identify problems, and reach (violent:) agreements, we are pretty sure we succeeded with WOI'00. The present report has collected (most of) the discussions that took place during and after the workshop, and the conclusions that were finally agreed.

Much work remains to be done in this area of object interoperability, but we hope that WOI'00, as its predecessor WOI'99, has provided the basis for concrete research activities in this field, and that following WOI's serve as a stable forum for discussion on these topics.

Before we finish, we would like to thank the ECOOP'2000 organization for giving us the opportunity to organize the workshop, especially to the Workshop Chairs, Sabine Moisan and Jacques Malenfant. Thanks also to all the contributing authors and to the referees who helped in choosing and improving the selected papers. Finally, we want to especially thank all WOI'00 attendees for their active and enthusiastic participation in the workshop, and for their contributions to this report. Many thanks to all for making WOI'00 a very enjoyable and productive experience.

References

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. on Software Engineering and Methodology*, July 1997.

[2] R. Bastide, O. Sy, and P. Palanque. A formal notation and tool for the engineering of CORBA systems. *Concurrency/TAPOS (Wiley). Special Issue "Selected papers from ECOOP'99"*, 2000. (to appear).
[3] S. Heiler. Semantic interoperability. *ACM Comp. Surveys*, 27(2):265–267, June 1995.
[4] J. Hernández, A. Vallecillo, and J. M. Troya, editors. *New Issues in Object Interoperability*. Universidad de Extremadura, Dept. Informática, 2000.
[5] D. Konstantas. Interoperation of object oriented applications. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*. Prentice-Hall, 1995.
[6] G. T. Leavens and M. Sitaraman, editors. *Foundations of Component-Based Systems*. Cambridge University Press, 2000.
[7] O. Nierstrasz. Regular types for active objects. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 99–121. Prentice-Hall, 1995.
[8] A. Vallecillo, J. Hernández, and J. M. Troya. Object interoperability. In *Object-Oriented Technology: ECOOP'99 Workshop Reader*, number 1743 in LNCS, pages 1–21. Springer-Verlag, 1999.
[9] P. Wegner. Interoperability. *ACM Comp. Surveys*, 28(1):285–287, March 1996.

Annex 1: List of Participants

Mehmet Aksit. University of Twente, The Netherlands (`aksit@cs.utwente.nl`)
Francisco Dominguez. Univ. Rey Juan Carlos, Spain (`f.dominguez@escet.urjc.es`)
Jun Han. Monash University, Australia (`jhan@monash.edu.au`)
Juan Hernández. Universidad de Extremadura, Spain (`juanher@unex.es`)
Duane Hybertson. The MITRE Coprporation, USA (`dhyberts@mitre.org`)
António Ravara. Technical University of Lisbon, Portugal (`amar@math.ist.utl.pt`)
Rick van Rein. University of Twente, The Netherlands (`vanrein@cs.utwente.nl`)
Ralf H. Reussner. University of Karlsruhe, Germany (`reussner@ira.uka.de`)
Antonio Ruiz. Universidad de Sevilla, Spain (`aruiz@lsi.us.es`)
Ousmane Sy. University Toulouse 1, France (`sy@univ-tlse1.fr`)
Bedir Tekinerdogan. University of Twente, The Netherlands (`bedir@cs.utwente.nl`)
Antonio Vallecillo. Universidad de Málaga, Spain. (`av@lcc.uma.es`)
Jack C. Wileden. University of Massachusetts, USA (`wileden@cs.umass.edu`)